Fall 10-12-2012

A Model for Improved Resource Profiling and Prediction for Mobile Applications

Adam Rehn

Jason Holdsworth

A Model for Improved Resource Profiling and Prediction for Mobile Applications

Adam Rehn, Jason Holdsworth
James Cook University
adam.rehn@my.jcu.edu.au, jason.holdsworth@jcu.edu.au

Abstract: Many mobile applications rely on the 'cloud's' vast computing power, but fail to exploit its full capability. Some application techniques facilitate this objective, but individual application requirements often present degrees of inflexibility. In this paper we propose a more flexible and dynamic computational offloading approach that enables deeper integration with mobile applications. We discuss how this is of benefit to mobile application developers and users.
Keywords: Computational offloading, mobile applications, profiling.

1. Introduction

Mobile applications have become big business. Statistics published in July 2011 by Apple show that total downloads from the iOS App Store, the most popular smartphone application repository, have exceeded 15 billion [7]. In 2012, Google reported that over 400 million Android devices had been activated [8]. The applications for these mobile platforms often rely on an internet connection to provide rich and dynamic user experiences. Hence, it is in application developers' best interests to maximise the benefits from this connection to ensure the richest and most robust user experience possible.

There are a number of benefits that mobile applications can receive when developers exploit the ability to communicate with remote servers. The most prominent technical benefits are extended battery life and significantly improved execution speed [4]. These benefits are desirable for all mobile applications, but can be technically complex to achieve because applications must account for changing hardware conditions at runtime. This problem is solved in the form of software systems known as computational offloading frameworks [4]. Although the development of computational offloading frameworks is a relatively mature field [9], there are still limitations imposed by the design of existing implementations. In this paper we propose a model to address what we perceive to be the most significant of these limitations.

This paper is structured as follows: the first part of the paper describes the function and design of existing computational offloading frameworks based on relevant literature. The second part of the paper describes the motivations and design of our proposed model for improving computational offloading frameworks. We conclude the paper by examining the advantages and limitations of our proposed model, and discuss ideas for future work.

2. Computational Offloading Frameworks

As modern mobile devices becoming increasingly more powerful, user expectations rise in tandem. However, mobile devices such as smartphones will always be restricted by limited or otherwise constrained resources, such as finite battery life, memory, and processing power. By comparison, traditional server hardware remains powerful and relatively unbounded. With the increasing ubiquity of cloud computing, a common technique known as *computational*

Proceedings of the Twelfth International Conference on Electronic Business, Xi'an, China, October 12-16, 2012, 184-190.

offloading is becoming extremely popular [4]. Computational offloading is a technique whereby a portion of an application's processing is offloaded and performed by a remote server in the cloud. When harnessed by a mobile application, computational offloading allows developers to effectively augment the abilities of any smartphone by several orders of magnitude [10].

A popular area of research has been the development of computational offloading frameworks [2] [3] [4] [6] [10]. These are low-level systems that sit as a layer beneath smartphone applications to facilitate the conditional offloading of their processing. These frameworks allow an application to adapt dynamically to changing runtime conditions, such as server availability and network bandwidth, and make offloading decisions to optimise performance and resource use [4]. At an abstract level, offloading frameworks function based on the interaction between two key components: the profiler and the solver. The relationship between these components is depicted in Figure 1.

The first component is the profiler. This component collects data on the resource usage and execution time of the application's methods [4]. Only methods that are candidates for offloading are profiled – methods that make use of resources unique to the client, such as GPS location, camera, or microphone, are ineligible for execution on the remote server [2].

The second component is the solver. The solver is the component responsible for making offloading decisions, based on the data collected from both environmental sensors and the profiler [1]. The solver supplies the profiler data to a prediction mechanism, which attempts the determine the resource usage and processing time for a given method based on the previously collected data [4].

Although the profilers in existing frameworks tend to be non-customisable, a number of frameworks provide mechanisms to allow developers to customise the behaviour of the solver [1] [5]. This allows application developers to modify the behaviour of the framework to better suit the unique characteristics of their applications, or even provide customisation options to the end user [1].

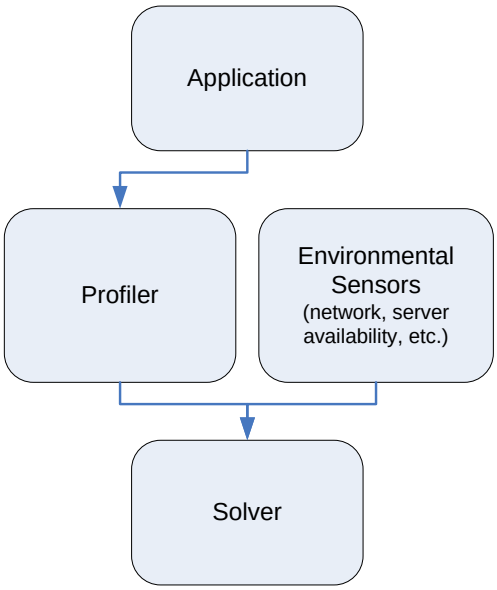

Figure 1: Data flow between key components of an offloading framework

3. Business Value of Offloading Frameworks

Computational offloading frameworks are designed to work with any mobile application, requiring developers to perform relatively little modification in order to integrate the framework into a newly developed mobile app [4] [2].

Utilising an offloading framework represents a significant saving in both development time and cost for mobile applications. Offloading frameworks can provide sophisticated and flexible offloading behaviour that would require a significant investment of time for developers to produce themselves. Those frameworks that provide support for behaviour customisation represent the best return on a developer's

investment, providing a way to optimise the framework for an individual application's needs in exchange for relatively little extra effort.

4. Existing Methodology

The solvers in existing frameworks utilise a history-based resource demand prediction mechanism [1] [2] [3] [4] [5] [6] [10], where resource demand and execution time is predicted based on data collected at runtime. The benefit of this online learning mechanism is that it allows offloading frameworks to adapt to actual runtime conditions as execution progresses, and it's device-agnostic – an important consideration given the sheer number of heterogeneous mobile devices in the modern market.

4.1 Granularity

The focus of an offloading framework's solver is to determine whether or not to offload a method. Methods are the individual functional units of an application. A method is comprised of an execution sequence of instructions which perform a single task. The execution flow of a method is influenced by two factors. The first factor is conditional evaluation, whereby different subsets of a method's instructions will be executed based on a condition. The second factor is repetition, whereby subsets of a method's instructions will be repeated multiple times, also based on a condition.

The profilers in existing implementations [1] [2] [3] [4] [5] [6] profile the resource usage of applications at a method level. Method-level profiling considers each method as a whole, ignoring the details of its execution flow. This approach is convenient due to its conceptual simplicity, and has a low processing overhead [4].

4.2 Shortcomings

In spite of its advantages, method-level resource profiling and prediction cannot account for the unique execution characteristics of individual methods in an application. This is because the resource demand of any given function may vary drastically, given different sets of input [10]. Processing complexity for certain types of data will also depend on the complexity of the data itself, in addition to its length [10]. These characteristics are highly application-specific, and may prove difficult to determine through runtime analysis [1].

5. Proposed Model

With the shortcomings of method-level profiling in mind, we proceeded to develop a new model to address these limitations. In the following sections we describe the goals and design of the proposed framework, followed by a discussion of its advantages and limitations.

5.1 Goals

In developing a model to improve profiling and prediction mechanisms we were guided by two goals, designed to address the shortcomings of existing method-level models:

Increase the accuracy of prediction by increasing method profiling granularity. Existing implementations consider each method as a whole, which limits their accuracy. We believe that the potentially varied nature of a method's execution cannot be captured unless the granularity of the profiler is increased.

Adapt to input data characteristics. The inability to account for the interaction between a method's inputs and its unique processing characteristics was the most significant limitation of method-level profiling identified in our study of existing implementations. Although this relationship is difficult to capture and express as an input to existing models, a new model can be designed with this consideration as an inherent factor. This goal instructed the design of both the pro-

filer and solver in the new model.

5.2 Model

To achieve our stated goals, we propose a model that significantly increases the granularity of both the profiling and prediction mechanisms. The key technique is to partition a method into blocks based on its control flow, as shown in Figure 2.

Profiling is performed based on each block in a method, instead of the method as a whole. In order to do so, profiler "probes" are inserted at the beginning of each block, which are then activated at runtime when the corresponding block is executed. If a block is not executed, its associated probe will not be activated. This allows the profiling runtime to determine exactly which execution paths have been invoked and correlate the collected resource usage and timing data accordingly.

To allow the prediction mechanism to take this fine-grained data into consideration, unique prediction functions need to be generated for each method. These prediction functions are based on the control flow of the corresponding method, and take the same input data. The prediction function accumulates predicted demand by mimicking the execution of the source method, based on the input data given. For example, if a block of the source method is only executed when the input data possesses a certain characteristic, the prediction function will only factor in the collected data for that block if the input data provided to it also possesses the required characteristic.

The combination of block-level probes and unique prediction functions for each method provide the mechanism by which the unique execution and processing characteristics of an application's methods can be captured and utilised. The interactions between the key elements of the model are depicted in Figure 3.

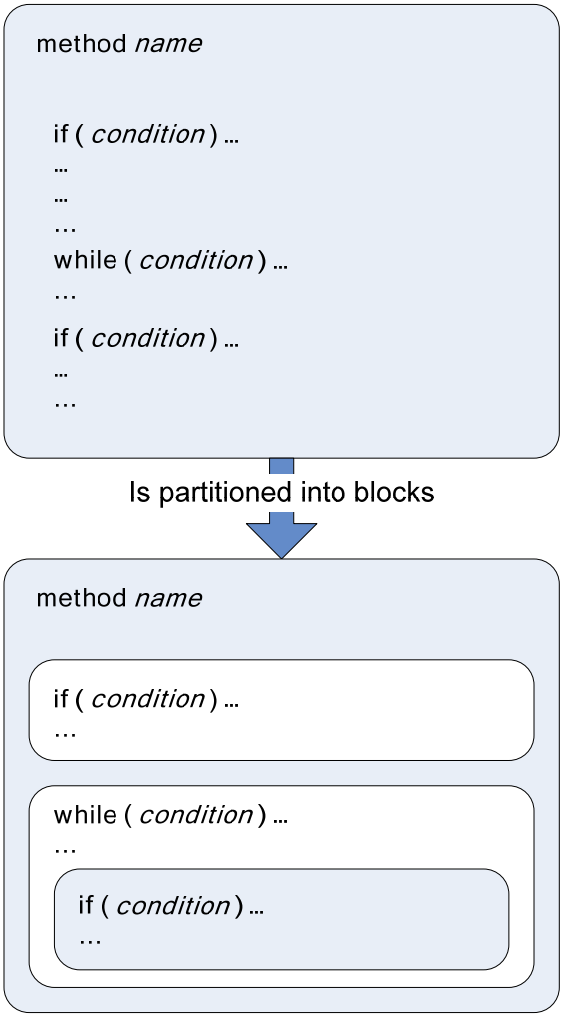

Figure 2: Breakdown of a method into blocks

The key elements of the proposed model are:

Source-to-Source Modification Tool. This tool analyses the source code of each method that has been marked as an offloading candidate. The tool then modifies the source code by inserting the profiler probes for each block. The static analysis performed by this tool is also used to generate the corresponding prediction function for each processed method.

Prediction Functions. Generated based on the analysis performed by the Source-To-Source Modification Tool, these functions predict the resource demand for their corresponding source method, based on the input data given to them.

Profiler Runtime. The runtime interacts with the profiler probes and resource monitors to collect resource usage and timing data for the invocations of methods.

The combination of block-level probes and unique prediction functions for each method provide the mechanism by which the unique execution and processing characteristics of an application's methods can be captured and utilised. The interactions between the key elements of the model are depicted in Figure 3.

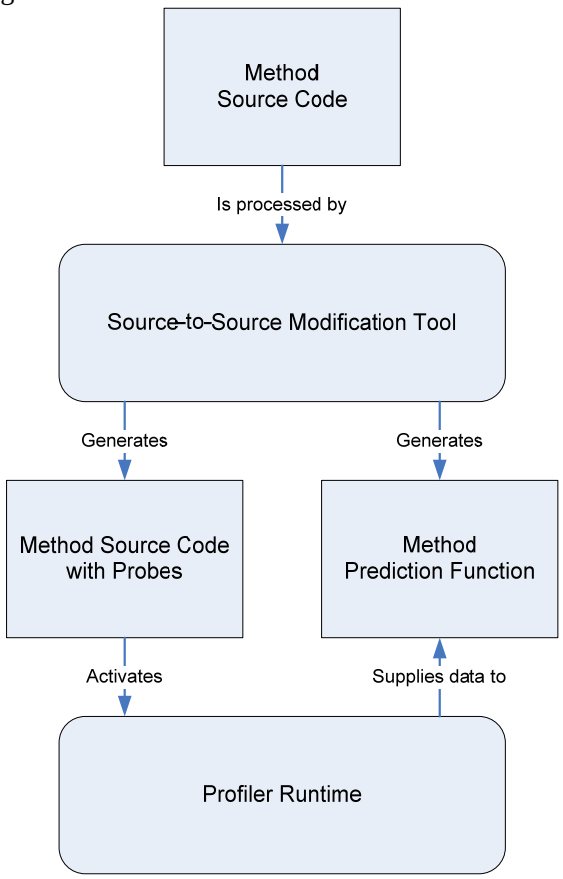

Figure 3: Interactions between key elements of the proposed model

The key elements of the proposed model are:

Source-to-Source Modification Tool. This tool analyses the source code of each method that has been marked as an offloading candidate. The tool then modifies the source code by inserting the profiler probes for each block. The static analysis performed by this tool is also used to generate the corresponding prediction function for each processed method.

Prediction Functions. Generated based on the analysis performed by the Source-To-Source Modification Tool, these functions predict the resource demand for their corresponding source method, based on the input data given to them.

Profiler Runtime. The runtime interacts with the profiler probes and resource monitors to collect resource usage and timing data for the invocations of methods.

5.3 Advantages

The primary advantage of the proposed model over existing implementations is that the unique execution characteristics of an application's methods can be captured, and subsequently utilised by the prediction mechanism. This allows an offloading framework to provide dynamic adaption on two fronts – both in response to the current environmental conditions and in response to the input data to the application.

5.4 Implications for research

We are currently working to integrate the proposed model into a prototype solver as a proof-of-concept. Once this is complete, we believe a logical next step would be to modify an existing offloading framework. If the source code for an existing framework could be obtained, it could be modified to utilise the proposed model.

In future, we believe that the developers of new computational offloading frameworks could benefit from the ideas outlined in our model. The proposed model is only one technique for capturing and utilising data on the unique execution behaviours of an application's methods. The underlying ideas of the model have the potential to provide numerous benefits to computational offloading frameworks. These

include increased flexibility, deeper integration with application code, and context-sensitive behaviour for profilers and solvers. The authors eagerly anticipate further improvements on the techniques presented here, as well as new approaches to achieve the same goals. The authors particularly look forward to new computational offloading frameworks that account for the unique execution characteristics of applications' methods.

5.5 Limitations

The primary limitation of the suggested model is the way in which demand prediction functions are designed. Because prediction functions mimic the execution flow of the source method, portions of the processing from the source method must be replicated – this is the key to determining which blocks will be executed based on the input data. This makes the model well-suited to methods which perform only light processing in their conditional evaluations, and have no dependencies between a conditional evaluation and heavy processing performed in a prior block. Methods which feature heavy conditional evaluation processing or feature such dependencies will be unsuitable for block-level profiling, since the prediction function would replicate this heavy processing and incur an unacceptably high overhead.

To mitigate this issue, we plan to look at ways to improve the Source-To-Source Modification Tool's static analysis methods so that dependencies between condition evaluations and processing in prior blocks can be identified. In the cases that a dependency is identified, the method could then be marked for use with method-level profiling instead. This can be easily facilitated by the insertion of a different set of profiler probes, and the generation of a simpler prediction function for the method.

6. Conclusion

Computational offloading is an increasingly important technique. Offloading allows mobile applications to fully exploit the capabilities of cloud computing. This maximises the benefits that an application receives from its connection to the internet. Computational offloading frameworks allow mobile applications to adapt dynamically to changing runtime conditions, to ensure the best user experience possible. We have identified key limitations in existing implementations of computational offloading frameworks which utilise method-level profiling and prediction. The primary limitation is an inflexibility in adapting to an application's individual requirements.

Based on the identified limitations of method-level profiling and prediction, we have proposed a model for a block-level profiling and prediction technique. This model allows an offloading framework to capture and utilise data on the unique execution characteristics of an application's methods. The utilisation of this data allows for more flexible and dynamic offloading behaviour. The ideas from the model will benefit future computational offloading frameworks' flexibility and integration with applications. Future work will continue to optimise and expand this model to maximise its performance.

The development of computational offloading frameworks is of benefit to all mobile application developers. There are a number of exciting opportunities to improve on the existing designs and implementations of offloading frameworks. Future frameworks utilising techniques similar to those of our proposed model will represent even greater value to mobile applications.

References

[1] Balan, R.K., Satyanarayanan, M., Young Park, S., & Okoshi, T. “Tactics based remote execution for mobile computing.” *Proceedings 1st international conference on Mobile systems, applica-*

tions and services, MobiSys '03, ACM. New York: NY, 2003, 273–286.

[2] Chun, B-G., Ihm, S., Maniatis, P., Naik, M., & Patti, A. "Clone Cloud: elastic execution between mobile device and cloud." *Proceedings 6th conference on Computer systems*, EuroSys '11, ACM, New York: NY, 2011, 301–314.

[3] Cidon, A., London, T.M., Katti, S., Kozyrakis, C., & Rosenblum, M. "MARS: adaptive remote execution for multi-threaded mobile devices." *Proceedings 3rd ACM SOSP Workshop on Networking, Systems, and Applications on Mobile Handhelds*, MobiHeld '11, ACM, New York: NY, 2011, 1:1–1:6.

[4] Cuervo, E., Balasubramanian, A., Cho, D-K., Wolman, A., Saroiu, S., Chandra, R., & Bahl, P. "MAUI: making smartphones last longer with code offload." *Proceedings 8th international conference on Mobile systems, applications, and services*, MobiSys '10, ACM, New York: NY, 2010, 49–62.

[5] Flinn, J., Young Park, S., & Satyanarayanan, M. "Balancing performance, energy, and quality in pervasive computing." *Proceedings 22nd International Conference on Distributed Computing Systems*, ICDCS '02, IEEE Computer Society, Washington: DC, 2002, 217–226.

[6] Imai, S., & Varela, C.A. "Light-weight adaptive task offloading from smartphones to nearby computational resources." *Proceedings 2011 ACM Symposium on Research in Applied Computation*, RACS '11, ACM, New York: NY, 2011, 146–152.

[7] Anon. "Apple's app store downloads top 15 billion." Apple Inc. Press Release, July 2011.

[8] Anon. "Google I/O 2012 keynote." Google Inc., June 2012.

[9] Noble, B.D.. Satyanarayanan, M., Narayanan, D.. Tilton, J.E., Flinn, J., & Walker, K.R. "Agile application-aware adaptation for mobility." *Proceedings 16th ACM symposium on Operating systems principles*, SOSP '97, ACM, New York: NY, 1997, 276–287.

[10] Ra, M-R., Sheth, A., Mummert, L., Pillai, P., Wetherall, D. & Govindan, R. "Odessa: enabling interactive perception applications on mobile devices." *Proceedings 9th international conference on Mobile systems, applications, and services*, MobiSys '11, ACM, New York: NY, 2011, 43–56.